Examples for Generating and Using Fragment Cache Identifiers

As described above, caching information is associated with each fragment that instructs caches how to cache that fragment. For static content, caching information is associated with each fragment. Dynamic content is generated by a template or program (JSP, CGI, etc.), and caching information would be associated with this template. This could be constant information, so that all fragments generated by the template would have the same values. Alternatively, the template could have code that determines the caching information, so that it can be different for each generated fragment based on some algorithm. In either case, a specific fragment has constant values.

A fragment can be defined as a portion of content that has been delimited for combination with another portion of content. A standardized fragment naming technique is used when implementing the present invention; the technique generates cache IDs in accordance with a technique that was described more formally above. This section describes the use of cache IDs through a series of examples further below, although a brief recap of the formation and determination of cache IDs is first provided.

A cache stores the fragment using a cache ID in some manner. Enough information should be included in the cache ID to make it unique among all applications using the cache. For example, a product ID alone might collide

with another store's product ID or with something else entirely. Since the URI path for a fragment typically has to address this same name scoping problem at least in part, it is convenient to include the URI path as part of the cache ID for a fragment.

The information content of a cache ID determines how widely or narrowly the fragment is shared, as shown in the following examples.

(A) If a user ID is included in a cache ID, then the fragment is used only for that user.

(B) If a shopper group ID is included in a cache ID, then the fragment is shared across all members of that shopper group.

(C) If no user ID or shopper group ID is included in a cache ID, then the fragment is shared across all users.

A Web application developer can specify the information content of a cache ID by a rule in the fragment's HTTP FRAGMENT header with a CACHEID directive that states what is included in the fragment's cache ID. A rule allows any URI query parameter or cookie to be appended to the URI path, or allows the full URI (including query parameters). The absence of a rule means do not cache. When multiple rules are used, the rules are tried in order of appearance. The first rule that works determines the cache ID. If no rule works, then the fragment is not cached. When a query parameter or cookie is included in the cache ID, it can be either required or optional, as follows.

(A) A required query parameter that is not present in the parent's request causes the rule to fail.

(B) A required cookie that is not present in the parent's request or in the result causes the rule to fail.

(C) An optional query parameter or cookie that is not present is not included in the cache ID.

A cache ID is case-sensitive except for those parts that some standard has declared case-insensitive. The HTTP specification states that a URI's protocol and host name are case-insensitive while the rest of the URI is case-sensitive including query parameter names. According to the specification "HTTP State Management Mechanism", RFC 2109, Internet Engineering Task Force, February 1997, cookie names are case-insensitive. A cache implementation can easily enforce this by transforming these case insensitive parts to a uniform case. The fragment caching technique of the present invention preferably makes query parameter values and cookie values case-sensitive.

With reference now to **Figures 11A-11H**, a series of diagrams are used to illustrate the manner in which the technique of the present invention constructs and uses unique cache identifiers for storing and processing fragments.

Referring to **Figure 11A**, all parent fragments at a site contain the same sidebar child fragment. The parent fragment is not specified in this scenario except that all parents contain the same sidebar fragment, so only the child fragment is at issue. The child fragment is logically qualified by its URI.

Since it is static content, its cache ID is the full URI. The cache ID rule would be:

Fragment: cacheid="URI"

In other words, the cache ID is the full URI including all query parameters. An example of the cache ID would be:

http://www.acmeStore.com/sidebar.html

Referring to **Figure 11B**, a product description page contains no embedded or child fragments, i.e. the page is the only fragment. It is logically qualified by the productID. The page URI has a productID query parameter. The page request has an encrypted userID cookie that is created by the Web application server during logon. The userID cookie allows user-specific state (shopping cart, user profile, etc.) to be associated with the user. The userID is used as a cookie rather than a query parameter because it may be used with almost every request, and it would be tedious for the Web application developer to put it in every link. The single cache ID rule for the product page could use the full URI as the cache ID, which includes the productID query parameter, so that it can be cached with the correct qualifications. For this single fragment page, the cache ID can be its URI. The cache ID rule would be:

Fragment: cacheid="URI"

In other words, the cache ID is the full URI including all query parameters. An example of the cache ID would be:

http://www.acmeStore.com/productDesc.jsp?productID=AT13394

Another way to specify the cache ID for this top-level fragment is the product ID used by the merchant, e.g., AT13394, which is a URI query parameter, plus the constant URI path to ensure uniqueness, e.g., http://www.acmeStore.com/productDesc. In this case, the cache ID rule would be:

Fragment: cacheid="(productId)"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path; and

(B) the name and value of the productID query parameter.

The lack of square brackets in the rule indicates that the productID parameter should exist. Otherwise, the rule fails, and the fragment will not be cached. An example of the cache ID would be:

http://www.acmeStore.com/productDesc.jsp_productID=AT13394

It should be noted again that the Web application developer specifies only the information content of a cache ID, not the exact format. The cache implementations can choose their own way to encode the specified information content in the cache ID. The above example uses simple concatenation with an underscore character ("_") as a separator delimiter. The Web application developer does not need to know this encoding.

Referring to **Figure 11C**, an extension of the product description scenario is provided. The price is now determined by which shopper group in which the user belongs, but the rest of the product description is

independent of shopper group. A parent product description fragment contains a child price fragment. The parent is logically qualified by the productID. The child is logically qualified by the productID and the groupID. The page URI has a productID query parameter. The page request has encrypted userID and groupID cookies. The groupID cookie is created by the Web application during logon based on the user profile. The groupID is made a cookie rather than a query parameter because it may be used with almost every request, and it would be tedious for the Web application developer to put it in every link.

The price should be in a separate child fragment included by the parent. The single cache ID rule for the parent fragment would be the same as in the product display scenario. The single cache ID rule for the child fragment would use the URI path along with the productID query parameter and groupID cookie, so that it can be cached with the correct qualifications. It should be noted that the cache ID does not include user ID because then the fragment could only be used by a single user instead of all users belonging to the same shopper group, thereby resulting in a much larger cache and more work to keep the cache updated. The cache ID rule would be:

Fragment: cacheid="(productID, [groupID])"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the productID query parameter; and

(C) the name and value of the groupID cookie if present in the request.

A comma separates the URI query parameters from cookies. The square brackets in the rule indicate that the cookie is optional. If this cookie is not present, the rule can still succeed, and the cache ID will not include the cookie name-value pair. This allows the merchant to have a no-group price as well as a price per group. An example of the cache ID would be:

http://www.acmeStore.com/productDesc.jsp_productID=AT13394_groupID=*@#!

Referring to **Figure 11D,** an extension of the shopper group scenario is provided. Support for multiple merchants has been added; for example, an application service provider (ASP) supports multiple merchants in the same Web application server using multiple languages. The parent product description fragment again contains a child price fragment. The parent is logically qualified by productID, merchantID, and languageID. The child is logically qualified by productID, groupID, languageID and merchantID. The page URI has productID and merchantID query parameters. The request has userID, groupID, and languageID cookies. The languageID cookie is created by the Web application during logon based on the user profile. The languageID is made a cookie rather than a query parameter because it is used with every request, and it would be tedious for the Web application developer to put it in every link.

The single cache ID rule for the parent fragment would use the URI path along with the productID and merchantID query parameters, and languageID cookie, so it

can be cached with the correct qualifications. The parent cache ID rule would be:

Fragment: cacheid="(productID merchantID, [languageID])"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the productID query parameter;

(C) the name and value of the merchantID query parameter; and

(D) the name and value of the languageID cookie if present in the request.

An example of the parent cache ID would be:

http://www.acmeMall.com/productDesc.jsp_productID=AT13394_merchantID=MyStore_languageID=eng

The single cache ID rule for the child fragment would use the URI path along with productID and merchantID query parameters, and groupID and optional languageID cookies, so it can be cached with the correct qualifications. The cache ID rule would be:

Fragment: cacheid="(productID merchantID,[groupID] [languageID])"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the productID query parameter;

(C) the name and value of the merchantID query parameter;

(D) the name and value of the groupID cookie if it is present in the request; and

(E) the name and value of the languageID cookie if it is present in the request.

An example of the cache ID would be:

http://www.acmeMall.com/productDesc.jsp_productID=AT13394_merchantID=MyStore_groupID=*@#!_languageID=eng

Referring to **Figure 11E**, an extension to the ASP and multiple languages scenario is provided. Support has been added for multiple ways to identify products. The parent product description fragment contains a child price fragment. The parent is logically qualified by product (there are two ways to specify this), languageID, and merchantID. The child is logically qualified by product, groupID, languageID, and merchantID. The product is identified either by the productID query parameter, or by partNumber and supplierNumber query parameters. The request has userID, groupID, and languageID cookies. The parent fragment would require two rules, which are specified as:

Fragment: cacheid=

"(productID merchantID, [languageID])

(partNumber supplierNumber merchantID, [languageID])"

The first rule is tried. If it succeeds, then it determines the cache ID. If it fails, the second rule is tried. If the second rule succeeds, then it determines the cache ID. If it fails, the fragment is not cached. The first rule means that the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the productID query parameter;

(C) the name and value of the merchantID query parameter; and

(D) the name and value of the languageID cookie if present in the request.

An example of the cache ID for the first rule would be:

http://www.acmeStore.com/productDesc.jsp_productID=AT13394_merchantID=MyStore_languageID=eng

The second rule means that the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the partNumber query parameter;

(C) the name and value of the supplierNumber query parameter;

(D) the name and value of the merchantID query parameter; and

(E) the name and value of the languageID cookie if present in the request.

An example of a cache ID for the second rule would be:

http://www.acmeStore.com/productDesc.jsp_partNumber=22984Z_supplierNumber=339001_merchantID=MyStore_languageID=eng

The child fragment requires two rules, which are specified as follows:

Fragment: cacheid=

"(productID merchantID, [groupID] [languageID])

(partNumber supplierNumber merchantID, [groupID] [languageID])"

The first rule is tried. If it succeeds, then it determines the cache ID. If it fails, then the second rule is tried. If the second rule succeeds, then it determines the cache ID. If the second rule fails, the fragment is not cached. The first rule means that the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the productID query parameter;

(C) the name and value of the merchantID query parameter;

(D) the name and value of the groupID cookie if it is present in the request; and

(E) the name and value of the languageID cookie if it is present in the request.

An example of a cache ID for the first rule would be:

http://www.acmeStore.com/productDesc.jsp_productID=AT13394_merchantID=MyStore_groupID=*@#!_languageID=eng

The second rule means that the cache ID is the following parts concatenated together:

(A) the URI path;

(B) the name and value of the partNumber query parameter;

(C) the name and value of the supplierNumber query parameter;

(D) the name and value of the merchantID query parameter;

(E) the name and value of the groupID cookie; and

(F) the name and value of the languageID cookie.

An example of a cache ID for the second rule would be:

http://www.acmeStore.com/productDesc.jsp_partNumber=22984Z_supplierNumber=339001_merchantID=MyStore_groupID=*@#!_language=eng

Referring to **Figure 11F**, an extension to the product description scenario using personalization is provided. A parent product description fragment contains a child personalization fragment. The parent fragment is logically qualified by the productID. The child fragment is logically qualified by the userID. The page URI has a productID query parameter. The request has a userID cookie.

The parent cache ID includes the productID query parameter. The cache ID rule for the parent fragment would be either of the following two cases:

Fragment: cacheid="URI"

In other words, the cache ID is the full URI with all query parameters. Another potential rule would be:

Fragment: cacheid="(productId)"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path; and

(B) the name and value of the productID query parameter.

It should be noted that even though the request for this page includes a userID cookie, it is not included in the cache ID for either fragment because the fragment is product-specific and not user-specific. If it were included, then this fragment would only be accessible by that user, resulting in a larger cache and more work to

keep the cache updated. An example of a cache ID would be:

http://www.acmeStore.com/productDesc.jsp_productID=AT13394

The child personalization fragment's cache ID includes a userID cookie. The child fragment's cache ID rule would be:

Fragment: cacheid="(, userId)"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path; and

(B) the name and value of the userID cookie.

An example of a cache ID would be:

http://www.acmeStore.com/personalization.jsp_userID=@($*!%

In this personalization example, the personalization fragments should be marked as private data, e.g., by using "Cache-Control: private".

Referring to **Figure 11G**, a parent stock watchlist fragment on a simple portal page contains multiple child stock quote fragments. The parent fragment also contains the user's name as a simple personalization. The parent is logically qualified by userID, i.e. the list of stock symbols is user-specific. The user name is logically qualified by the userID. Each child is logically qualified by its stock symbol, i.e. a stock's value is not user-specific. The page URI contains no query parameters. The request has a userID cookie.

The top-level fragment contains a required user-specific list of stock quotes. The top-level fragment's URI contains no query parameters. The

top-level fragment's cache ID includes an encrypted cookie named userID. The cache ID rule would be:

Fragment: cacheid="(, userId)"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path; and

(B) the name and value of the userID cookie.

An example of a cache ID would be:

http://www.acmeInvest.com/stockList.jsp_userID=@($*!%

For each of the stock quote fragments, the cache ID includes the "symbol" parameter. The cache ID rule would be the full URI or the URI path plus the stockSymbol query parameter:

Fragment: cacheid="(stockSymbol)"

In other words, the cache ID is the following parts concatenated together:

(A) the URI path; and

(B) the name and value of the symbol query parameter.

An example of a cache ID would be:

http://www.acmeInvest.com/stockQuote.jsp_stockSymbol=IBM

This scenario can be modified to use the FOREACH feature; the stock quote fragments would not change, but the parent fragment can be highly optimized. There is only one static top-level fragment. A stockSymbols cookie would be used whose value is a blank-separated list of stock symbols for the user. There would be only one parent fragment for all users that is quite static, which contains a FRAGMENTLINK tag whose FOREACH attribute

would name the stockSymbols cookie. This dynamically generates a simple FRAGMENTLINK for each stock symbol whose SRC attribute is the same as the SRC of the FRAGMENTLINK containing the FOREACH attribute with the stock symbol added as a query parameter. Because this parent fragment is the same for all users, it can be cached with the correct qualifications with a single cache rule that uses its URI as the cache ID, which has no query parameters, as follow:

Fragment: cacheid="URI"

The stockSymbols cookie contains all the user-specific information for the parent fragment and travels with the page request, so it satisfies the parent's logical userID qualification.

A userName cookie whose value is the user's name would be used in a FRAGMENTLINK tag for the simple personalization whose SRC attribute identifies the userName cookie. This fragment is not cached since it can easily be generated from the userName cookie. The userName cookie contains all the user-specific information for this fragment and travels with the page request, so it satisfies the parent's logical userID qualification.

The single cache ID rule for the child fragment uses its URI for the cache ID so that it can be cached with the correct qualifications, as follows:

Fragment: cacheid="URI"

In this stock watchlist scenario, when the FOREACH feature is not being used, the top-level stock watchlist fragments would be marked private, e.g., by using "Cache-Control: private". When the FOREACH feature is

used, then there is only one top-level fragment that is shared, so it is not marked private.

Referring to **Figure 11H**, the example depicts a scenario that is similar to a personalized portal page, such as myYahoo!. A first-level portal fragment contains multiple mid-level topic fragments, such as stocks, weather, sports, each of which contains multiple leaf item fragments. The parent fragment also contains the user's name. The top-level portal fragment is logically qualified by the userID, i.e. the list of topics is user-specific. The user name is logically qualified by the userID. The mid-level topics fragment is logically qualified by the topicID and userID, i.e. the list of items in the topic is user-specific. The leaf item fragment is logically qualified by the itemID, i.e. an item's value is not user-specific. The page URI contains no query parameters. The page request has a userID cookie. Through the use of the FOREACH feature, the parent fragment can be highly optimized.

Using the FOREACH feature, a topics cookie (created during logon based on user profile) would be used whose value is a blank-separated list of topicIDs for that user. There would be only one parent fragment for all users that is quite static, containing a FRAGMENTLINK tag whose FOREACH attribute would name the topics cookie. This dynamically generates a simple FRAGMENTLINK for each topicID, whose SRC attribute is the same as the SRC of the FRAGMENTLINK containing the FOREACH attribute with the topicID appended as a query parameter. Because this parent fragment is the same for all users, it can be cached with the correct qualifications with a single

cache rule that uses its URI as the cache ID, which has no query parameters, as follows:

Fragment: cacheid="URI"

The topics cookie contains all the user-specific information for the parent fragment and travels with the page request, so it satisfies the parent's logical userID qualification. A userName cookie whose value is the user's name would be used in a FRAGMENTLINK for the simple personalization whose SRC attribute identifies the userName cookie. This fragment is not cached since it can easily be generated from the userName cookie. The userName cookie contains all the user-specific information for this fragment and travels with the page request, so it satisfies the parent's logical userID qualification.

There is a topic fragment for each topic. Because of the FOREACH feature, each of the topic fragments can be highly optimized. For each topic, a cookie (created during logon based on user profile) would be used whose value is a blank-separated list of itemIDs for that user and topic. For each topic, there would be only one topic fragment for all users that is quite static containing a FRAGMENTLINK whose FOREACH attribute would name the corresponding cookie for that topic. This dynamically generates a simple FRAGMENTLINK for each itemID whose SRC attribute is the SRC of the FRAGMENTLINK containing the FOREACH attribute with the itemID added as a query parameter (the topicID query parameter is already there). Because each topic fragment is the same for all users, it can be cached with the correct qualifications with a single cache rule that uses its URI as the cache ID,

which has its topicID as a query parameter. The topics cookie contains all the user-specific information for the topic fragment and travels with the page request, so it satisfies the topic fragment's logical userID qualification.

The URI for each item fragment contains its topicID and itemID as query parameters. The single cache ID rule for each item fragment uses its URI for the cache ID, so it can be cached with the correct qualifications.

Examples for the Specification of FRAGMENTLINK Tags

Referring again to the sidebar example in **Figure 11A**, a single FRAGMENTLINK would be placed in the page instead of the sidebar and in the same location where the sidebar is desired, such as:

```
{fragmentlink
 src="http://www.acmeStore.com/sidebar.html"}
```

Referring again to the shopper group example in **Figure 11C**, a single FRAGMENTLINK would be located where the price would be, such as:

```
{fragmentlink
 src="http://www.acmeStore.com/productPrice.jsp"}
```

The URI that is constructed for a particular price fragment would look as follows:

http://www.acmeStore.com/productPrice.jsp?productID=AT13394

The request for the fragment includes all of the parent's query parameters, i.e. "productId", and cookies, i.e. "groupId", so that they are available during the execution of productPrice.jsp in the application server.

Referring again to the personalization example in **Figure 11F**, the top-level fragment would include a FRAGMENTLINK located where the personalized fragment is desired, such as:

{fragmentlink
src="http://www.acmeStore.com/personalization.jsp"}

The URI that is constructed for a particular user-specific personalization fragment would look like as follows:

http://www.acmeStore.com/personalization.jsp?productID=AT13394

The request for the fragment includes all of the parent's query parameters (ie, "productId") and cookies (ie, "userId"). During the execution of personalization.jsp, the "userId" cookie is used but the "productId" query parameter is ignored.

Referring again to the stock watchlist example in **Figure 11G**, the top-level fragment would include a variable number of FRAGMENTLINK tags that depends on how many stock quotes that the user wanted. Each FRAGMENTLINK tag would be located where the stock quotes would be. Each would look as follows:

{fragmentlink
src="http://www.acmeInvest.com/stockQuote.jsp?symbol=IBM"}

The URI that is constructed for a particular stock quote fragment would look as follows:

http://www.acmeInvest.com/stockQuote.jsp?symbol=IBM

This scenario can be modified to use the FOREACH feature; the variable number of FRAGMENTLINK tags are replaced by a single FRAGMENTLINK tag with the FOREACH

attribute specifying the name of a cookie (stocks) whose value is a blank-separated list of stock symbol parameters:

{fragmentlink
src="http://www.acmeInvest.com/stockQuote.jsp"
foreach="stocks"}

If the value of the cookie named "stocks" was

symbol=IBM symbol=CSCO symbol=DELL

then this would be equivalent to the following set of FRAGMENTLINK tags:

{fragmentlink
src="http://www.acmeInvest.com/stockQuote.jsp?symbol=IBM"}

{fragmentlink
src="http://www.acmeInvest.com/stockQuote.jsp?symbol=CSCO"}

{fragmentlink
src="http://www.acmeInvest.com/stockQuote.jsp?symbol=DELL"}

Referring again to the full portal example in **Figure 11H**, the FOREACH feature can be used for a single static top-level fragment that would be shared by all users. The userName in the top-level fragment would be included using the following FRAGMENTLINK that identifies the userName cookie, which contains the user's name:

{fragmentlink src="cookie://userName"}

The top-level fragment would also have a FRAGMENTLINK tag whose FOREACH attribute identifies the topics cookie, which contains that user's list of topics:

{fragmentlink
src="http://www.acmePortal.com/portalPage.jsp"
foreach="topics"}

This cookie contains a list of topicIDs. For a topics cookie whose value is the following:

topic=stocks topic=weather topic=tv

the above FRAGMENTLINK containing the FOREACH attribute would generate the following simple FRAGMENTLINKS:

{fragmentlink
src="http://www.acmePortal.com/portalPage.jsp?topic=stocks"}

{fragmentlink
src="http://www.acmePortal.com/portalPage.jsp?topic=weather"}

{fragmentlink
src="http://www.acmePortal.com/portalPage.jsp?topic=tv"}

Each of the dynamically generated SRC attributes locates a fragment that handles the specified topic.

The implementation of "portalPage.jsp" in the Web application server acts as a dispatcher that calls a fragment based on the query parameters. No parameter returns the top-level fragment. A "topic=stocks" query parameter returns the stocks topic fragment. Using the stocks topic fragment as an example, and again using the FOREACH feature, the stocks topic fragment contains a FRAGMENTLINK whose FOREACH attribute identifies a stocks cookie, which contains that user's list of stock symbols for that topic:

{fragmentlink
src="http://www.stockQuotes.com/stockQuote.jsp"
foreach="stocks"}

An exemplary use of this would be to generate rows of a table with a row for each stock symbol in the stocks cookie. For a "stocks" cookie whose value is

symbol=IBM symbol=DELL symbol=CSCO

the above FRAGMENTLINK containing the FOREACH attribute would dynamically generate the following FRAGMENTLINKS:

{fragmentlink
src="http://www.stockQuotes.com/stockQuote.jsp?symbol=IBM"}
{fragmentlink
src="http://www.stockQuotes.com/stockQuote.jsp?symbol=DELL"}
{fragmentlink
src="http://www.stockQuotes.com/stockQuote.jsp?symbol=CSCO"}

Examples of Passing Data From Parent Fragment to Child Fragment

A fragment should be as self-contained as possible. There are two reasons for this. The first reason is that good software engineering dictates that software modules should be as independent as possible. The number and complexity of contracts between modules should be minimized, so that changes in one module are kept local and do not propagate into other modules. For example, an application might get data in a parent module and pass this data into a child module that formats it. When this

is done, there has to be a contract describing what the data is and how it is to be passed in. Any change in what data is needed by the child module requires changes to both modules. Instead, if the child module gets its own data, then the change is kept local. If there is a need to make either module independent of how its data is obtained, or the code that obtains its data is the same in several modules, then a separate data bean and a corresponding contract can be used to accomplish either of these requirements. However, adding yet another contract between the parent and child modules is only added complexity without accomplishing anything.

The second reason that a fragment should be as self-contained as possible is that to make caching efficient, the code that generates a fragment should be self-contained. In the above example, if the parent module gets all the data for the child module and passes it into the child, then the child itself only does formatting. With this dependency between modules, if the data needed by the child module becomes out of date, then both the parent and child have to be invalidated and generated again. This dependency makes caching of the separate fragments much less effective. A fragment that is shared by multiple parents complicates both of the above problems.

The JSP programming model allows data to be passed between JSPs via request attributes or session state. For nested fragments, the request attribute mechanism does not work because the parent and child JSPs may be retrieved in different requests to the application server. Also, the session state mechanism may not work

if the parent and child can be executed in different sessions. Instead, any information that should be passed should use URI query parameters or cookies. Even a complex data structure that was passed from parent to child using request attributes could still be passed by serializing it and including it as a query parameter in the URI in the FRAGMENTLINK tag's SRC attribute.

Even when fragments get their own data, there is still a need to pass some control data between them. Referring to the above examples again, in the sidebar scenario, no data is passed from the top-level fragments to the sidebar. In the shopper group scenario, the top-level product-description fragment needs to know the product ID, and the child group-product specific price needs both the product ID and the shopper group ID. The product ID is supplied by the external request. The shopper group ID is generated by the application using the user ID, both of which are generated at logon. Both the product ID and the shopper group ID should be passed through the product description fragment to the price fragment. All URI query parameters and cookies are automatically passed to the child fragment.

In the personalization scenario, the top-level product description fragment needs to know the product ID, and the child personalization fragment needs to know the user ID. Both of these parameters are supplied by the external request, so the user ID should be passed through the product description fragment to the personalization fragment. This is done by passing the cookie named "userId" on to the child fragment.

In the stock watchlist scenario, the top-level stock watchlist fragment needs to know the user ID cookie, and each of the child stock quote fragments need to know the stock symbol. The stock symbols and the FRAGMENTLINK tags that contain them are generated as part of the top-level stock watchlist fragment. The stock symbol should be passed to the stock quote fragment. This is done by putting the stock symbol as a query parameter of the URI in the SRC attribute of the FRAGMENTLINK.

Examples of FRAGMENTLINK tags and FRAGMENT headers

With reference now to **Tables 1A-1C**, a set of HTML and HTTP statements are shown for the sidebar example discussed above. Both fragments within this scenario are static. The parent top-level fragment would be a JSP because it contains another fragment using a "jsp:include" and because cache control information needs to be associated with the parent fragment. The child sidebar fragment is also a JSP because caching control information needs to be associated with it, but it does not contain any JSP tags.

Table 1A shows a JSP including HTML statements for the top-level fragment that contains the sidebar fragment.

```
{html}
{head}
    {title}A page containing a side bar.{/title}
{/head}
{body}
    {!-- Add the side bar. --}
    {jsp:include page="/sidebar.html"}
    {p}This is the rest of the body.
{/body}
{/html}
```

TABLE 1A

Table 1B shows the HTTP output that would be generated by a Web application server for the top-level fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 246
Content-Type: text/html
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="URL"
Cache-Control: max-age=600
Fragment: contains-fragments

{html}
{head}
    {title}A page containing a side bar.{/title}
{/head}
{body}
{%-- Add the side bar --%}
   {fragmentlink src="http://www.acmeStore.com/sidebar.html"}

    ... This is the rest of the body ...

{/body}
{/html}
```

TABLE 1B

Table 1C shows the HTTP output that would be generated by a Web application server for the sidebar fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="URL"
Cache-Control: max-age=6000

{html}
{body}
{p}This is the side bar body.
{/body}
{/html}
```

TABLE 1C

With reference now to **Tables 2A-2D**, a set of HTML and HTTP statements are shown for the shopper group example discussed above. Both fragments within this scenario are dynamic. A JSP is used for the top-level fragment that contains the product-group-specific price fragment. The child fragment is also a JSP because it contains business application logic for obtaining the appropriate price.

Table 2A shows a JSP containing HTML statements for the top-level product description fragment that contains the child fragment.

```
{html}
{head}
      {title}Product description.{/title}
{/head}
{body}
{h1} Product with Shopper Group. {/h1}
{%@ page    language="java"
            import="com.acmeStore.databeans.*"
%}
{%
      // Add the product description.
      ProductSGDataBean databean = new ProductSGDataBean();
      databean.setProductId(request.getParameter("productId"));
      databean.execute();
      out.println("{p}Product id is " + databean.getProductId());
%}
      {%-- Add the price --%}
      {jsp:include page="/groupPrice.jsp"}
{/body}
{/html}
```

TABLE 2A

Table 2B shows the HTTP output that would be generated by a Web application server for the product description fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(productId)"
Cache-Control: max-age=600
Fragment: contains-fragments

{html}
{head}
      {title}Product description.{/title}
{/head}
{body}
{h1} Product with Shopper Group. {/h1}
      ...The formatted product descriptions would be here...
      {fragmentlink src="http://www.acmeStore.com/groupPrice.jsp"}
{/body}
{/html}
```

TABLE 2B

Table 2C shows a JSP containing HTML statements for the child product-group-specific price fragment.

```
{html}
{body}
{%@ page language="java"
         import="com.acmeStore.databeans.*" %}
{%  // Get the groupId from its cookie.
    Cookie[] cookies = request.getCookies();
    String groupId = null;
    for (int i = 0; i { cookies.length; i++) {
        if (cookies[i].getName().equals("groupId")) {
            groupId = cookies[i].getValue();
        }
    }
    // Get the price.
    GroupPriceDataBean databean = new GroupPriceDataBean();
    databean.setGroupId(groupId);
    databean.execute();
    String price = databean.getPrice();
    out.println("{p}Price is " + price); %}
{/body}
{/html}
```

TABLE 2C

Table 2D shows the HTTP output that would be generated by a Web application server for the product-group-specific price fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: private
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(productId, groupId)"
Fragment: dependencies="http://www.acmeStore.com_groupid=*@#!"

{html}
{body}
    Price is $24.99
{/body}
{/html}
```

TABLE 2D

With reference now to **Tables 3A-3D**, a set of HTML and HTTP statements are shown for the personalization example discussed above. Both fragments within this scenario are dynamic. A JSP that generates the top-level product fragment contains a single user-specific personalization fragment. The child fragment is also a JSP because it contains business application logic for obtaining the appropriate personalization data for the user.

Table 3A shows a JSP containing HTML statements for the top-level product description fragment that contains the child fragment.

```
{html}
{head}
{title}Product description.{/title}
{/head}
{body}
{%@ page    language="java"
      import="com.acmeStore.databeans.*,com.acmeStore.formatters.*"
%}
{%
      // Add the product description.
      ProductDataBean databean = new ProductDataBean();
      databean.setProductId(request.getParameter("productId"));
      databean.execute();
      ProductFormatter productFormatter = new ProductFormatter();
      out.println(productFormatter.format(databean));
%}
{%-- Add the personalization --%}
{jsp:include page="/personalization.jsp"}
{/body}
{/html}
```

TABLE 3A

Table 3B shows the HTTP output that would be generated by a Web application server for the product description fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(productId)"
Cache-Control: max-age=600
Fragment: contains-fragments

{html}
{head}
      {title}Product description.{/title}
{/head}
{body}
{h1} Product with Shopper Group. {/h1}
      ... The formatted product descriptions would be here ...
      {fragmentlink src="http://www.acmeStore.com/personalization.jsp"}
{/body}
{/html}
```

TABLE 3B

Table 3C shows a JSP containing HTML statements for the child user-specific fragment.

```
{html}
{body}
{%@ page language="java"
      import="com.acmeStore.databeans.*" %}
{%  // Get the userId from the userId cookie.
      Cookie[] cookies = request.getCookies();
      String userId = null;
      for (int i = 0; i { cookies.length; i++) {
            if (cookies[i].getName().equals("userId")) {
                  userId = cookies[i].getValue(); } }
      "dependencies=\"http://www.acmeStore.com/userId=@($*!%\"");
      response.addHeader("Fragment", "cacheid=\"(, userId)\"");
      // this one depends on userId:
      response.addHeader("Fragment",
            "dependencies=\"http://www.acmeStore.com/userId=" +
            userId + "\"");
      // Create the personalization.
      PersonalizationDataBean databean =
                  new PersonalizationDataBean();
      databean.setUserId(userId);
      databean.execute();
      String personalization = databean.getPersonalization();
      out.println(personalization); %}
{/body}
{/html}
```

TABLE 3C

Table 3D shows the HTTP output that would be generated by a Web application server for the child fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: private
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(, userId)"
Fragment: dependencies="http://www.acmeStore.com_userId=@($*!%"

{html}
{body}
    ... The personalization would be here ...
{/body}
{/html}
```

TABLE 3D

With reference now to **Tables 4A-4F**, a set of HTML and HTTP statements are shown for the stock watchlist example discussed above. Both fragments within this scenario are dynamic.

Table 4A shows a JSP that generates the top-level stock watchlist fragment that contains multiple stock quote fragments. The "jspext:cookie" tag displays the user name that is in a cookie named "userName". This example dynamically generates a variable number of "RequestDispatcher.include" method invocations, each generating a FRAGMENTLINK tag in the output.

```
{html}
{head}
{title}Stock watch list.{/title}
{/head}
{body}
{%@ page    language="java"
            import="com.acmeInvest.databeans.*"
%}
{%
      // Get the userId from the userId cookie.
      Cookie[] cookies = request.getCookies();
      String userId = null;
      for (int i = 0; i { cookies.length; i++) {
            if (cookies[i].getName().equals("userId")) {
                  userId = cookies[i].getValue();
            }
      }
%}
{table border}
      {tr}
            {th colspan=2 align=center}
            {jspext:cookie name="userName"}'s Stock Watch List:
            {/th}
      {/tr}
            {tr}
                  {th align=center}Symbol{/th}
                  {th align=center}Price{/th}
            {/tr}
{%
      // Add the stock watch list rows to the table.
      StockListDataBean databean = new StockListDataBean();
      databean.setUserId(userId);
      databean.execute();
      String[] symbols = databean.getStockSymbolList();
      for (int i = 0; i { symbols.length; i++) {
            String url = "/stockQuote.jsp?stockSymbol=" + symbols[i];
            ServletContext servletContext = getServletContext();
            RequestDispatcher requestDispatcher =
            servletContext.getRequestDispatcher("/stockQuote.jsp");
            requestDispatcher.include(request, response);
      }
%}
{/table}
{/body}
{/html}
```

TABLE 4A

Table 4B shows the HTTP output that would be generated by a Web application server for the stock watchlist fragment.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: private
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(, userId)"
Fragment: contains-fragments

{html}
{body}
{table border}
      {tr}
            {th colspan=2 align=center}
            {fragmentlink src="cookie://userName"}'s Stock Watch List:
            {/th}
      {/tr}
      {tr}
            {th align=center}Symbol{/th}
            {th align=center}Price{/th}
      {/tr}
      {fragmentlink
      src="http://www.acmeInvest.com/stockQuote.jsp?symbol=IBM"}
      {fragmentlink
      src="http://www.acmeInvest.com/stockQuote.jsp?symbol=CSCO"}
      {fragmentlink
      src="http://www.acmeInvest.com/stockQuote.jsp?symbol=DELL"}
{/table}
{/body}
{/html}
```

TABLE 4B

Table 4C shows a JSP that generates the top-level stock watchlist fragment that incorporates a FOREACH attribute.

```
{html}
{head}
{title}Stock watch list.{/title}
{/head}
{body}
{%@ page    language="java"
            import="com.acmeInvest.databeans.*" %}
{table border}
      {tr}
            {th colspan=2 align=center}
            {jspext:cookie name="userName"}'s Stock Watch List:
            {/th}
      {/tr}
      {tr}
            {th align=center}Symbol{/th}
            {th align=center}Price{/th}
      {/tr}
      {jspext:include page="/stockQuote.jsp" foreach="stocks"}
{/table}
{/body}
{/html}
```

TABLE 4C

Table 4D shows the HTTP output that would be generated by a Web application server for the top-level stock watchlist fragment that incorporates a FOREACH attribute.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 246
Content-Type: text/html
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: contains-fragments
Fragment: cacheid="URL"
Cache-Control: max-age=600

{html}
{head}
      {title}Stock watch list.{/title}
{/head}
{body}
      {table border}
            {tr}
            {th colspan=2 align=center}
            {fragmentlink src="cookie://userName"}'s Stock Watch List:
            {/th}
            {/tr}
            {tr}
                  {th align=center}Symbol{/th}
                  {th align=center}Price{/th}
            {/tr}
            {fragmentlink
                        src="http://www.acmeInvest.com/stockQuote.jsp"
                        foreach="stocks"}
      {/table}
{/body}
{/html}
```

TABLE 4D

Table 4E shows a JSP that generates the individual stock quote.

```
{html}
{body}
{%@ page    language="java"
            import="com.acmeInvest.databeans.*"
%}
{%
      // Add the stock quote.
      StockQuoteDataBean databean = new StockQuoteDataBean();
      String symbol = request.getParameter("symbol");
      databean.setStockSymbol(symbol);
      databean.execute();
      String quote = databean.getStockQuote();
      String rtn =
            "{tr}" +
            "{td align=center}" + symbol + "{/td}" +
            "{td align=right}" + quote + "{/td}" +
            "{/tr}";
      out.println(rtn);
%}
{/body}
{/html}
```

TABLE 4E

Table 4F shows the HTTP output that would be generated by a Web application server for a symbol query parameter "IBM".

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2002 17:04:04 GMT
Server: IBM_HTTP_Server/1.3.6.2 Apache/1.3.7-dev (Unix)
Last-Modified: Wed, 11 Apr 2001 21:05:09 GMT
ETag: "b7-d8d-3ad4c705"
Accept-Ranges: bytes
Content-Length: 82
Content-Type: text/html
Cache-Control: private
Cache-Control: no-cache fragmentrules
Pragma: no-cache
Fragment: cacheid="(, userId)"
Cache-Control: max-age=1200

{html}
{body}
    {tr}
        {td align=center}IBM{/td}
        {td align=right}$112.72{/td}
    {/tr}
{/body}
{/html}
```

TABLE 4F

Conclusion

The advantages of the present invention should be apparent in view of the detailed description of the invention that is provided above. A fragment caching technique can be implemented within a cache management unit that may be deployed in computing devices throughout a network such that the cache management units provide a distributed fragment caching mechanism.

A FRAGMENT header is defined to be used within a network protocol, such as HTTP; the header associates metadata with a fragment for various purposes related to the processing and caching of a fragment. For example, the header is used to identify whether either the client, server, or some intermediate cache has page assembly abilities. The header also specifies cache ID rules for forming a cache identifier for a fragment; these rules may be based on a URI for the fragment, or the URI path and some combination of the query parameters from the URI, and cookies that accompany the request. In addition, the header can specify the dependency relationships of fragments in support of host-initiated invalidations.

The FRAGMENTLINK tag is used to specify the location in a page for an included fragment which is to be inserted during page assembly or page rendering. A FRAGMENTLINK tag is defined to contain enough information to either find the linked fragment in a cache or to retrieve it from a server. Cache ID rules are used both when a fragment is being stored in the cache and when processing a source identifier from a request to find the

fragment within a cache. To find the fragment in the cache, the cache ID rules that are associated with the fragment's URI path are used to determine the cache ID. The rules allow a high degree of flexibility in forming a cache ID for a fragment without having to deploy a computer program that forces a standard implementation for cache ID formation. Multiple cache ID rules may be used. The cache ID rules allow a cache ID to be a full URI for a fragment or the URI and a combination of query parameters or cookies. This scheme allows the same FRAGMENTLINK to locate different fragments depending on the parent fragment's query parameters and cookies; for example, a user ID cookie in the request for a product description page could be used to form the cache ID for a personalization fragment.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that some of the processes associated with the present invention are capable of being distributed in the form of instructions in a computer readable medium and a variety of other forms, regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include media such as EPROM, ROM, tape, paper, floppy disc, hard disk drive, RAM, and CD-ROMs and transmission-type media, such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration but is not

intended to be exhaustive or limited to the disclosed embodiments. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiments were chosen to explain the principles of the invention and its practical applications and to enable others of ordinary skill in the art to understand the invention in order to implement various embodiments with various modifications as might be suited to other contemplated uses.